ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

Deep Convolutional Neural Networks with Sequentially Semiseparable Weight Matrices

Matthias Kissel and Klaus Diepold

Technical University of Munich - Chair of Data Processing
Arcisstr. 21 - 80333 Munich - Germany

Abstract. Modern Convolutional Neural Networks (CNNs) comprise millions of parameters. Therefore, the use of these networks requires high computing and memory resources. We propose to reduce these resource requirements by using structured matrices. For that, we replace weight matrices of the fully connected classifier part of several pre-trained CNNs by Sequentially Semiseparable (SSS) Matrices. By that, the number of parameters in these layers can be reduced drastically, as well as the number of operations required for evaluating the layer. We show that the combination of approximating the original weight matrices with SSS matrices followed by gradient-descent based training leads to the best prediction results (compared to just approximating or training from scratch).

1 Introduction

Modern Convolutional Neural Networks (CNNs) consist of many layers and millions of parameters. By that, they are able to achieve remarkable results in image-based problem domains, like image recognition [1, 2]. However, as the number of parameters increases, so does the computational effort required for deploying the network. This is especially a problem for applications targeting mobile devices or embedded hardware like microcontrollers. For these applications, the use of modern CNN architectures is often not possible due to insufficient computing and / or memory resources.

Deep CNNs are often designed similarly. First, there is a feature extractor part, which consists of convolutional and pooling layers. After the feature extractor part, the activations are flattened and put into the classifier part. The classifier usually consists of fully-connected feed-forward layers. A large part of the parameters of the network is typically located in the classifier part, since the parameters in the convolutional filters are shared. As a result, evaluating the classifier part involves performing large matrix-vector multiplications.

Our goal is to reduce the resources needed in the classifier part of deep CNNs. For that, we focus on the last (i.e. fully-connected) layer of the network. Propagating information through this layer requires $\mathcal{O}(nm)$ operations for a weight matrix $W \in \mathcal{R}^{n \times m}$. This order of magnitude can be reduced to the subquadratic domain if the weight matrix of the layer has a specific structure. Particularly, we are interested in using Sequentially Semiseparable (SSS) matrices as weight matrices in neural networks. This matrix structure typically arises when describing linear time-varying systems [3].

Our contribution is two-fold. First, we investigate the effect of replacing the last weight matrix of several state-of-the art CNN models with a SSS matrix. By

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

that, we can analyze the trade-off between number of parameters and prediction accuracy of the overall recognition model. Second, we study the influence of the method used to bring the structure into the neural network. Here, we compare the achieved prediction performance of different approaches like approximating the weight matrix, training an SSS weight matrix from scratch, or the combined approach of approximation and training.

The rest of this paper is organized as follows. We first give an overview over approaches in literature, which use structured matrices in the context of neural networks. Subsequently, we introduce the methods we use to bring the SSS structure into the neural network. In Section 4, we present and discuss our experimental results. Finally, we draw a conclusion.

2 Literature Review

Several approaches in literature proposed the use of structured matrices in neural networks. In this context, matrices of low displacement rank are often used [4]. Prominent representatives of this structure class are Toeplitz matrices, which are connected to CNNs. Other approaches focused on Toeplitz-like weight matrices [5], or trained the operator matrices together with the low-rank components end-to-end in a neural network [6]. Moreover, it has been shown that the universal approximation theorem holds for neural networks with weight matrices of low displacement rank [7].

Other matrix structures used in neural networks are hierarchical matrices [8] and products of sparse matrices [9]. For example, Fan et al. [10] proposed to use hierarchical matrices in neural networks. Later, they extended their approach to $\mathcal{H}^2$ matrices [11]. Several authors proposed to use products of sparse matrices (particularly butterfly matrices) in neural networks [9, 12, 13, 14]. The idea here is that the product of sparse matrices is not sparse in general. Therefore, many dense matrices can be represented as a product of sparse matrices.

3 Methods

Our goal is to replace a weight matrix $W \in \mathcal{R}^{n \times m}$ from a trained neural network with an SSS matrix $\hat{W}$. The SSS matrix has the form

$$\hat{W} = D + C(I - ZA)^{-1}ZB + G(I - Z^T E)^{-1}Z^T F. \tag{1}$$

Here, I is the identity matrix and Z is a down-shift matrix containing ones on the subdiagonal and zeros everywhere else. A, B, C, D, E, F and G are block-diagonal matrices, where each matrix contains p sub-matrices

$$A = diag([A_1, \ldots, A_p]) \tag{2}$$

(B, C, D, E, F and G matrices respectively). This structure naturally arises when describing time-varying systems [3], where matrices $A, \ldots, G$ explain the behavior of a system. For example, C maps the state of the system to the

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

output, and D maps the inputs of the system to the outputs. Note that the dimensions of the A_k, B_k, C_k, D_k, E_k, F_k and G_k matrices can change for different $k = 1, \ldots, p$. This is due to the fact that input, state and output dimensions might change over time.

We explore two approaches for replacing a given weight matrix with an SSS matrix. The first approach starts from a randomly initialized SSS matrix, which is trained using *Backpropagation through States* [15] (this is a data-driven approach). In contrast, no training data is required for the second approach. Instead, the original weight matrix is approximated using a model order reduction method. For both approaches, suitable dimensions for $A_k, \ldots, G_k$ need to be found. We set these dimensions with the aim to achieve a uniform distribution of the input and output dimensions in the SSS representation. This means, that for a given weight matrix $W \in \mathcal{R}^{n \times m}$, which is to be approximated with an SSS matrix with p stages, the resulting input dimensions $dim(u_k)$ are

$$dim(u_k) = \begin{cases} floor(\frac{m}{p}) + 1 & for \quad k \leq m - floor(\frac{m}{p})p, \\ floor(\frac{m}{p}) & else \end{cases} \tag{3}$$

(output dimensions analogously). The dimension of the states d_k is fixed to be constant for all k ($d_k = d$ for all k). We treat d as a hyper parameter to control the number of parameters available in the SSS matrix.

We use *Backpropagation through States* in order to train SSS weight matrices. The key idea of the algorithm is to derive the training loss $\mathcal{L}$ with respect to the parameters of the structure, *not* with respect to the entries of the resulting weight matrix $\hat{W}_{i,j}$. This results in gradients of the form

$$\frac{\delta \mathcal{L}}{\delta A_k}, \frac{\delta \mathcal{L}}{\delta B_k}, \frac{\delta \mathcal{L}}{\delta C_k}, \frac{\delta \mathcal{L}}{\delta D_k}, \frac{\delta \mathcal{L}}{\delta E_k}, \frac{\delta \mathcal{L}}{\delta F_k}, \frac{\delta \mathcal{L}}{\delta G_k}. \tag{4}$$

We compute these gradients using automatic differentiation as provided by the pytorch machine learning framework[1]. The other steps of the training procedure remain the same as in standard neural network training.

In order to approximate a given weight matrix with an SSS matrix, we use a model order reduction method [3, 16]. This is based on the standard approach for finding a balanced state-space realization for a given transfer operator (which is the original weight matrix in our case). As part of the realization algorithm, the Hankel matrices $\mathcal{H}_i$ of the operator are decomposed into observability and controllability matrices ($\mathcal{O}_i$ and $\mathcal{C}_i$ respectively) using the Singular Value Decomposition

$$\mathcal{H}_i = U_i S_i V_i^T, \quad \mathcal{O}_i = U_i \sqrt{S_i}, \quad \mathcal{C}_i = \sqrt{S_i} V_i^T. \tag{5}$$

At this step, we cut out the smallest singular values until the realization has the desired state dimension (d in our case). By that, we obtain a realization $\hat{W}$, which performs similar to the original weight matrix W, but with a reduced amount of parameters. This procedure is called balanced model reduction for time-invariant systems [3].

[1] https://pytorch.org/

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

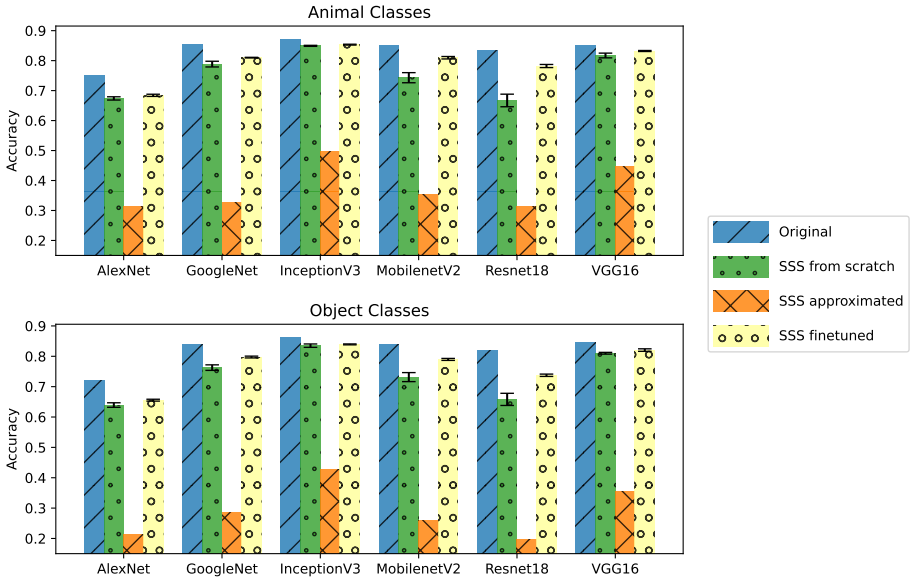

Fig. 1: Approximation with subsequent training led to the best results for all models (the resulting SSS matrix comprises 20% of the parameters).

4 Results

We conduct experiments with pretrained deep convolutional neural networks obtained from pytorch, namely *AlexNet*, *VGG16*, *ResNet18*, *InceptionV3*, *MobilenetV2*, and *GoogleNet*. The models are pretrained on the ImageNet 2012 dataset for image recognition [17]. For our experiments, we selected two subsets of images from the overall Imagenet dataset. Each subset comprises 100 classes from the original dataset (animals and objects), whereas each class comes with approximately 1000 training images and 50 validation images. By that, we can compare the effects on two distinct datasets for several models. We report the mean and standard deviation of the accuracy on the ImageNet validation set. This set has not been used in our training procedure (but it might have been used for pretraining the models).

We replace the weight matrix of the last, fully-connected layer with an SSS matrix. For this, we compare three approaches: Approximating the weight matrix, approximation followed by training, and training the sequentially semiseparable matrix from scratch (after random initialization). Our code used for conducting the experiments can be found online[2].

Replacing the original weight matrix with an SSS matrix obtained from balanced model reduction consistently led to bad prediction accuracy for the resulting model in all experiments. However, the combined approach of approximation

[2]https://github.com/MatthiasKi/structurednets

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

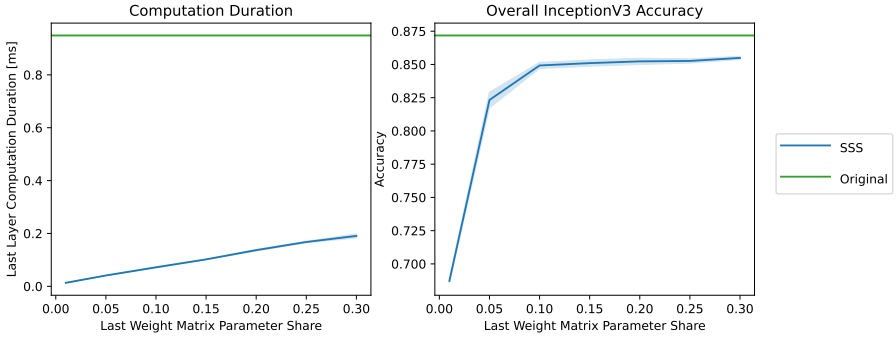

Fig. 2: Evaluating the last layer of the InceptionV3 model requires less time after replacing the weight matrix with an SSS matrix. However, the prediction accuracy of the InceptionV3 model also decreases.

followed by training led to better results than training a similar SSS matrix from scratch. The achieved final accuracy is lower than when using the original weight matrix, whereas the gap between the original accuracy and the accuracy of the model with SSS matrix depends on the model. For some models, the combined approach of approximating and training the SSS weight matrices achieved very good results, yielding a good trade-off between reduction in parameters and reduction of accuracy in these models. These results are depicted in figure 1.

Besides the accuracy of the final model, we are also interested in the duration required for running inference. For that, we compared the time required for evaluating the last layer of the InceptionV3 model (depicted in Figure 2). This speed comparison is implemented in C and executed on a single CPU core (of an Intel Core i-7-8750H CPU with 2.20 GHz). The time required for evaluating the matrix-vector multiplication increases with the number of parameters in the SSS matrix. For all investigated parameter shares the required computation time is significantly lower after replacing the original weight matrix with an SSS matrix.

5 Conclusion

We analyzed the effect of replacing weight matrices in the dense layers of deep CNNs with SSS matrices. The resulting modified layers require significantly less parameters to be stored, and can be evaluated much faster than the original layers. This is due to the structure of the SSS matrix, which facilitates efficient matrix-vector multiplication with subquadratic order of operations.

Our results showed that there is a trade-off between reducing the number of parameters and decrease in prediction accuracy. The performance depends on the number of parameters in the SSS matrix and the model at hand. We conclude that there is a lot of potential in the approach of optimizing trained neural networks by introducing SSS matrices.

ESANN 2022 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium) and online event, 5-7 October 2022, i6doc.com publ., ISBN 978287587084-1. Available from http://www.i6doc.com/en/.

References

[1] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[2] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[3] Patrick Dewilde and Alle-Jan Van der Veen. *Time-varying systems and computations*. Springer Science & Business Media, 1998.

[4] Victor Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.

[5] Vikas Sindhwani, Tara N Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 3088–3096, 2015.

[6] Anna T Thomas, Albert Gu, Tri Dao, Atri Rudra, and Christopher Ré. Learning compressed transforms with low displacement rank. *Advances in neural information processing systems*, 2018:9052, 2018.

[7] Liang Zhao, Siyu Liao, Yanzhi Wang, Zhe Li, Jian Tang, and Bo Yuan. Theoretical properties for neural networks with weight matrices of low displacement rank. In *international conference on machine learning*, pages 4082–4090. PMLR, 2017.

[8] Wolfgang Hackbusch. *Hierarchical matrices: algorithms and analysis*, volume 49. Springer, 2015.

[9] Tri Dao, Nimit Sohoni, Albert Gu, Matthew Eichhorn, Amit Blonder, Megan Leszczynski, Atri Rudra, and Christopher Ré. Kaleidoscope: An efficient, learnable representation for all structured linear maps. In *International Conference on Learning Representations*, 2020.

[10] Yuwei Fan, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical matrices. *Multiscale Modeling & Simulation*, 17(4):1189–1213, 2019.

[11] Yuwei Fan, Jordi Feliu-Faba, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núnez. A multiscale neural network based on hierarchical nested bases. *Research in the Mathematical Sciences*, 6(2):1–28, 2019.

[12] Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *International Conference on Machine Learning*, pages 1517–1527. PMLR, 2019.

[13] Luc Giffon, Stéphane Ayache, Hachem Kadri, Thierry Artières, and Ronan Sicre. Psm-nets: Compressing neural networks with product of sparse matrices. 2021.

[14] Nir Ailon, Omer Leibovitch, and Vineet Nair. Sparse linear networks with a fixed butterfly structure: theory and practice. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, volume 161, pages 1174–1184. PMLR, 2021.

[15] Matthias Kissel, Martin Gottwald, Biljana Gjeroska, Philipp Paukner, and Klaus Diepold. Backpropagation through states: Training neural networks with sequentially semiseparable weight matrices. *Proceedings of the 21st EPIA Conference on Artificial Intelligence*, 2022.

[16] Matthias Kissel, Sven Gronauer, Mathias Korte, Luca Sacchetto, and Klaus Diepold. Exploiting structures in weight matrices for efficient real-time drone control with neural networks. *Proceedings of the 21st EPIA Conference on Artificial Intelligence*, 2022.

[17] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.